

# Filter

Filter A comprehensive filter component that provides both single and multi-selection modes, grouped filter options, and flexible positioning. Built with accessibility in mind and designed for complex data filtering scenarios in data tables, search interfaces, and content management systems.

How to use

```
■ import { AavaFilterComponent } from "@aava/play-core";
```

Note : The Filter component is standalone and includes all necessary dependencies for checkbox, icon, and common modules.

Basic Usage

```
■ Simple filter implementation with multi-selection support and grouped options.
```

Angular Preview Code

```
< aava-filter size = " md " title = " Filter Products " [filterGroups] = " filterGroups " [showBadge] = " true " (filterChange) = " onFilterChange($event) " (clearAll) = " onClearAllFilters() " ></ aava-filter >< div class = " product-card " *ngFor = " let product of filteredProducts " >< h4 > {{ product.name }} </ h4 >< p > Category: {{ product.category }} </ p >< p class = " price " >${{ product.price }} </ p ></ div > filterGroups = [ { id : 'category' , title : 'Category' , options : [ { id : 'electronics' , label : 'Electronics' , value : 'electronics' } , { id : 'clothing' , label : 'Clothing' , value : 'clothing' } , { id : 'books' , label : 'Books' , value : 'books' } , { id : 'home' , label : 'Home & Garden' , value : 'home' } ] , } , { id : 'price' , title : 'Price Range' , options : [ { id : 'under-50' , label : 'Under $50' , value : { min : 0 , max : 50 } } , { id : '50-100' , label : '$50 - $100' , value : { min : 50 , max : 100 } } , { id : '100-200' , label : '$100 - $200' , value : { min : 100 , max : 200 } } ] , } , products = [ { id : 1 , name : 'Laptop' , category : 'electronics' , price : 999 } , { id : 2 , name : 'Smartphone' , category : 'electronics' , price : 699 } , { id : 3 , name : 'T-Shirt' , category : 'clothing' , price : 25 } , { id : 4 , name : 'Jeans' , category : 'clothing' , price : 45 } , { id : 5 , name : 'Novel' , category : 'books' , price : 15 } , { id : 6 , name : 'Cookbook' , category : 'books' , price : 35 } , { id : 7 , name : 'Coffee Table' , category : 'home' , price : 299 } , { id : 8 , name : 'Garden Chair' , category : 'home' , price : 89 } ] ; filteredProducts = [ ... this . products ] ; activeFilters : Record < string , FilterOption [ ] > = { } ; onFilterChange ( filters : Record < string , FilterOption [ ] > ) : void { console . log ( 'Filter changed:' , filters ) ; this . activeFilters = filters ; this . applyFilters ( ) ; } onClearAllFilters ( ) : void { console . log ( 'All filters cleared' ) ; this . activeFilters = { } ; this . filteredProducts = [ ... this . products ] ; } private applyFilters ( ) : void { this . filteredProducts = this . products . filter ( ( product ) => { // Category filter if ( this . activeFilters [ 'category' ] && this . activeFilters [ 'category' ] . length > 0 ) { const categoryMatch = this . activeFilters [ 'category' ] . some ( ( f : FilterOption ) => f . value === product . category ) ; if ( ! categoryMatch ) return false ; } // Price filter if ( this . activeFilters [ 'price' ] && this . activeFilters [ 'price' ] . length > 0 ) { const priceMatch = this . activeFilters [ 'price' ] . some ( ( f : FilterOption ) => { const priceRange = f . value as PriceRange ; return ( product . price >= priceRange . min && product . price <= priceRange . max ) ; } ) ; if ( ! priceMatch ) return false ; } return true ; } ) ; } getActiveFiltersCount ( ) : number { let count = 0 ; Object . values ( this . activeFilters ) . forEach ( ( group ) => { if ( Array . isArray ( group ) ) { count += group . length ; } } ) ; return count ; } getSelectedCategories ( ) : string { const categories = this . activeFilters [
```

```

'category' ] ; if ( ! categories || categories . length === 0 ) return 'None' ; return categories . map ( ( c : FilterOption ) => c . label ) . join ( ', ' ) ; } getSelectedPriceRanges ( ) : string { const prices = this . activeFilters [ 'price' ] ; if ( ! prices || prices . length === 0 ) return 'None' ; return prices . map ( ( p : FilterOption ) => p . label ) . join ( ', ' ) ; } Multi-Selection Mode ■ Advanced multi-selection capabilities with grouped filters and clear all functionality. Angular Preview Code < aava-filter size = " md " title = " Filter Users " [filterGroups] = " filterGroups " [showBadge] = " true " [showApplyButton] = " true " (filterChange) = " onFilterChange($event) " (clearAll) = " onClearAllFilters() " (apply) = " onApplyFilters($event) " > </ aava-filter > < table > < thead > < tr > < th > Name </ th > < th > Status </ th > < th > Role </ th > < th > Department </ th > < th > Experience </ th > </ tr > </ thead > < tbody > < tr *ngFor = " let user of filteredUsers " > < td > {{ user.name }} </ td > < td > < span class = " status-badge " [ngClass] = "' status-' + user.status " > {{ user.status }} </ span > </ td > < td > {{ user.role }} </ td > < td > {{ user.department }} </ td > < td > {{ user.experience }} </ td > </ tr > </ tbody > </ table > filterGroups = [ { id : 'status' , title : 'Status' , options : [ { id : 'active' , label : 'Active' , value : 'active' } , { id : 'inactive' , label : 'Inactive' , value : 'inactive' } , { id : 'pending' , label : 'Pending' , value : 'pending' } , { id : 'archived' , label : 'Archived' , value : 'archived' } , ] , } , { id : 'role' , title : 'Role' , options : [ { id : 'admin' , label : 'Admin' , value : 'admin' } , { id : 'user' , label : 'User' , value : 'user' } , { id : 'moderator' , label : 'Moderator' , value : 'moderator' } , { id : 'guest' , label : 'Guest' , value : 'guest' } , ] , } , { id : 'department' , title : 'Department' , options : [ { id : 'it' , label : 'IT' , value : 'IT' } , { id : 'hr' , label : 'HR' , value : 'HR' } , { id : 'finance' , label : 'Finance' , value : 'Finance' } , { id : 'marketing' , label : 'Marketing' , value : 'Marketing' } , { id : 'sales' , label : 'Sales' , value : 'Sales' } , ] , } , { id : 'experience' , title : 'Experience Level' , options : [ { id : 'junior' , label : 'Junior (0-2 years)' , value : 'junior' } , { id : 'mid' , label : 'Mid-level (3-5 years)' , value : 'mid' } , { id : 'senior' , label : 'Senior (6-10 years)' , value : 'senior' } , ] , } , ] ; users = [ { id : 1 , name : 'John Doe' , status : 'active' , role : 'admin' , department : 'IT' , experience : 'senior' } , { id : 2 , name : 'Jane Smith' , status : 'active' , role : 'user' , department : 'HR' , experience : 'mid' } , { id : 3 , name : 'Bob Johnson' , status : 'inactive' , role : 'moderator' , department : 'Finance' , experience : 'senior' } , { id : 4 , name : 'Alice Brown' , status : 'active' , role : 'user' , department : 'Marketing' , experience : 'junior' } , { id : 5 , name : 'Charlie Wilson' , status : 'pending' , role : 'user' , department : 'IT' , experience : 'mid' } , { id : 6 , name : 'Diana Davis' , status : 'active' , role : 'admin' , department : 'HR' , experience : 'expert' } , { id : 7 , name : 'Evan Miller' , status : 'archived' , role : 'guest' , department : 'Sales' , experience : 'junior' } , { id : 8 , name : 'Fiona Garcia' , status : 'active' , role : 'moderator' , department : 'IT' , experience : 'senior' } , ] ; filteredUsers = [ ... this . users ] ; activeFilters : Record < string , FilterOption [ ] > = { } ; onFilterChange ( filters : Record < string , FilterOption [ ] > ) : void { console . log ( 'Filter changed:' , filters ) ; this . activeFilters = filters ; this . applyFilters ( ) ; } onClearAllFilters ( ) : void { console . log ( 'All filters cleared' ) ; this . activeFilters = { } ; this . filteredUsers = [ ... this . users ] ; } onApplyFilters ( filters : Record < string , FilterOption [ ] > ) : void { console . log ( 'Filters applied:' , filters ) ; this . activeFilters = filters ; this . applyFilters ( ) ; } private applyFilters ( ) : void { this . filteredUsers = this . users . filter ( ( user ) => { // Status filter if ( this . activeFilters [ 'status' ] && this . activeFilters [ 'status' ] . length > 0 ) { const statusMatch =

```

```

this . activeFilters [ 'status' ] . some ( ( f : FilterOption ) => f . value === user . status ) ; if ( ! statusMatch ) return false ; } // Role filter if ( this . activeFilters [ 'role' ] && this . activeFilters [ 'role' ] . length > 0 ) { const roleMatch = this . activeFilters [ 'role' ] . some ( ( f : FilterOption ) => f . value === user . role ) ; if ( ! roleMatch ) return false ; } // Department filter if ( this . activeFilters [ 'department' ] && this . activeFilters [ 'department' ] . length > 0 ) { const deptMatch = this . activeFilters [ 'department' ] . some ( ( f : FilterOption ) => f . value === user . department ) ; if ( ! deptMatch ) return false ; } // Experience filter if ( this . activeFilters [ 'experience' ] && this . activeFilters [ 'experience' ] . length > 0 ) { const expMatch = this . activeFilters [ 'experience' ] . some ( ( f : FilterOption ) => f . value === user . experience ) ; if ( ! expMatch ) return false ; } return true ; } ) ; } getActiveFiltersCount ( ) : number { let count = 0 ; Object . values ( this . activeFilters ) . forEach ( ( group ) => { if ( Array . isArray ( group ) ) { count += group . length ; } } ) ; return count ; } getFilterSummary ( ) : string { const summaries : string [ ] = [ ] ; if ( this . activeFilters [ 'status' ] ? . length ) { summaries . push ( ` ${ this . activeFilters [ 'status' ] . length } status(es)` ) ; } if ( this . activeFilters [ 'role' ] ? . length ) { summaries . push ( ` ${ this . activeFilters [ 'role' ] . length } role(s)` ) ; } if ( this . activeFilters [ 'department' ] ? . length ) { summaries . push ( ` ${ this . activeFilters [ 'department' ] . length } department(s)` ) ; } if ( this . activeFilters [ 'experience' ] ? . length ) { summaries . push ( ` ${ this . activeFilters [ 'experience' ] . length } experience level(s)` ) ; } return summaries . length > 0 ? summaries . join ( ',' ) : 'No filters applied' ; } }

■ Multi-Selection Features ■ Grouped Filters : Organize filters into logical groups with titles
Multi-Select Groups : Support for both single and multi-selection within groups
Clear All : Bulk clear functionality for all selected filters
Active Count Badge : Visual indicator of active filter count
Apply Button : Optional apply button for controlled filter application
Single-Selection Mode ■ Streamlined single-selection interface optimized for quick filtering decisions
Angular Preview
Code < aava-filter [singleSelection] = " true " title = " Sort By " [filterGroups] = " sortFilterGroups "
[showClearAll] = " true " [showApplyButton] = " false " size = " md " (singleSelectionChange) = " onSortChange($event) "
> </ aava-filter > < div class = " product-card " *ngFor = " let product of sortedProducts; let i = index " >
< span class = " order-number " > #{{ i + 1 }} </ span > < h4 > {{ product.name }} < h4 > < p > Category: {{ product.category }} < p > < p class = " price " > ${{ product.price }} < p > < p class = " date " > {{ product.date | date : "shortDate" }} < p > < / div >
sortFilterGroups = [ { id : "sortBy" , title : "Sort By" , options : [ { id : "name-asc" , label : "Name (A-Z)" , value : "name_asc" } , { id : "name-desc" , label : "Name (Z-A)" , value : "name_desc" } ] , { id : "price-low" , label : "Price (Low to High)" , value : "price_asc" } , { id : "price-high" , label : "Price (High to Low)" , value : "price_desc" } ] , { id : "newest" , label : "Newest First" , value : "date_desc" } , { id : "oldest" , label : "Oldest First" , value : "date_asc" } ] , products = [ { id : 1 , name : 'Laptop' , category : 'electronics' , price : 999 , date : '2024-01-15' } , { id : 2 , name : 'Smartphone' , category : 'electronics' , price : 699 , date : '2024-01-10' } , { id : 3 , name : 'T-Shirt' , category : 'clothing' , price : 25 , date : '2024-01-20' } , { id : 4 , name : 'Jeans' , category : 'clothing' , price : 45 , date : '2024-01-18' } , { id : 5 , name : 'Novel' , category : 'books' , price : 15 , date : '2024-01-12' } , { id : 6 , name : 'Cookbook' , category : 'books' , price : 35 , date : '2024-01-14' } , { id : 7 , name : 'Coffee Table' , category : 'home' , price : 299 , date : '2024-01-16' } , { id : 8 ,

```

```
name : 'Garden Chair' , category : 'home' , price : 89 , date : '2024-01-19' , } , { id : 9 , name : 'Running Shoes' , category : 'sports' , price : 120 , date : '2024-01-17' , } , { id : 10 , name : 'Yoga Mat' , category : 'sports' , price : 45 , date : '2024-01-13' , } , ] ; sortedProducts = [ ... this . products ] ; selectedSort : FilterOption | null = null ; selectedCategory : FilterOption | null = null ; selectedStatus : FilterOption | null = null ; onSortChange ( selection : Record < string , FilterOption | null > ) : void { const sortOption = selection [ 'sortBy' ] ; this . selectedSort = sortOption ; console . log ( 'Sort changed:' , sortOption ) ; if ( sortOption ) { this . applySorting ( sortOption . value ) ; } } private applySorting ( sortBy : string ) : void { this . sortedProducts = [ ... this . products ] ; switch ( sortBy ) { case 'name_asc' : this . sortedProducts . sort ( ( a , b ) => a . name . localeCompare ( b . name ) ) ; break ; case 'name_desc' : this . sortedProducts . sort ( ( a , b ) => b . name . localeCompare ( a . name ) ) ; break ; case 'price_asc' : this . sortedProducts . sort ( ( a , b ) => a . price - b . price ) ; break ; case 'price_desc' : this . sortedProducts . sort ( ( a , b ) => b . price - a . price ) ; break ; case 'date_desc' : this . sortedProducts . sort ( ( a , b ) => new Date ( b . date ) . getTime ( ) - new Date ( a . date ) . getTime ( ) ) ; break ; case 'date_asc' : this . sortedProducts . sort ( ( a , b ) => new Date ( a . date ) . getTime ( ) - new Date ( b . date ) . getTime ( ) ) ; break ; } } getSelectedSortLabel ( ) : string { return this . selectedSort ? this . selectedSort . label : 'None' ; } getSelectedCategoryLabel ( ) : string { return this . selectedCategory ? this . selectedCategory . label : 'None' ; } getSelectedStatusLabel ( ) : string { return this . selectedStatus ? this . selectedStatus . label : 'None' ; }
```

**Single-Selection Features**

- Radio-like Behavior** : Only one option can be selected per group
- Auto-close** : Panel automatically closes after selection
- Visual Feedback** : Clear selection indicators with check icons
- Simplified Interface** : No checkboxes, clean click-to-select interaction

**Configuration**

- Flexible configuration and sizing options** for different layout requirements.
- Angular Preview Code**
- Features**

**Selection Modes**

**Multi-Selection**

**Default mode** with checkbox-based selection

**Single-Selection** : Radio-like behavior with auto-close

**Group-Level Control** : Individual groups can override selection behavior

**Mixed Modes** : Support for different selection types per group

**User Experience**

- Visual Feedback** : Active state indicators and selection badges
- Keyboard Navigation** : Full keyboard support for accessibility
- Responsive Design** : Adapts to different screen sizes
- Smooth Animations** : CSS transitions for panel open/close
- State Persistence** : Maintains selection state during interactions

**Filter Management**

- Grouped Organization** : Logical grouping of related filter options
- Clear All** : Bulk clear functionality for all filters
- Apply Control** : Optional apply button for controlled submission

**Active Count** : Real-time display of active filter count

**Position Flexibility** : Left or right-aligned panels

**API Reference**

- Inputs**
- Property Type** Default Description size FilterSize 'md' Size variant (sm, md, lg, xl) title string 'Filter' Title displayed on the filter button
- filterGroups** FilterGroup[] [] Array of filter groups with options
- showClearAll** boolean true Whether to show clear all button
- showApplyButton** boolean false Whether to show apply button
- isOpen** boolean false Whether the filter panel is open
- position** 'left' | 'right' Position of the filter panel
- maxHeight** string '400px' Maximum height of the filter panel
- width** string 'auto' Width of the filter panel
- disabled** boolean false Whether the filter is disabled
- class** string " Additional CSS classes
- singleSelection** boolean false Enable single-selection mode
- showBadge** boolean true Whether to

show active filter count badge Outputs ■ Event Type Description filterChange EventEmitter<{ [groupId: string]: FilterOption[] }> Emitted when filters change (multi-select) clearAll EventEmitter<void> Emitted when clear all is clicked apply EventEmitter<{ [groupId: string]: FilterOption[] }> Emitted when apply button is clicked toggleFilter EventEmitter<boolean> Emitted when filter panel is toggled singleSelectionChange EventEmitter<{ [groupId: string]: FilterOption | null }> Emitted when selection changes (single-select) Methods ■ Method Parameters Return Description toggleFilterPanel() None void Toggle the filter panel open/close onOptionChange() groupId: string, option: FilterOption, isChecked: boolean void Handle option selection change onSingleOptionClick() groupId: string, option: FilterOption void Handle single-selection option click clearAllFilters() None void Clear all selected filters applyFilters() None void Apply current filter selection getActiveFiltersCount() None number Get count of active filters getSizeClasses() None string Get CSS classes for current size getCheckboxSize() None 'sm' | 'md' | 'lg' Get appropriate checkbox size for current size getSize() None number Get appropriate icon size for current size isSelected() groupId: string, option: FilterOption boolean Check if option is selected Properties ■ Property Type Description selectedFilters { [groupId: string]: FilterOption[] } Current multi-selection state (private) singleSelectedFilters { [groupId: string]: FilterOption | null } Current single-selection state (private) Interfaces ■ FilterOption ■ interface FilterOption { id : string | number ; label : string ; value : any ; selected ? : boolean ; } FilterGroup ■ interface FilterGroup { id : string ; title : string ; options : FilterOption [ ] ; multiSelect ? : boolean ; } FilterSize ■ type FilterSize = "sm" | "md" | "lg" | "xlg" ; Design Tokens & Theming ■ AAVA Play Filter uses semantic design tokens for all surfaces, spacing, radius, and motion. The component exposes scoped override tokens for fine-tuning appearance while maintaining design system consistency. Available Design Tokens for Filter ■ Trigger Button Tokens ■ Token Purpose Default Value --filter-background-primary Primary background color Theme-based --filter-background-hover Hover background color Theme-based --filter-border Border style Theme-based --filter-border-hover Hover border color Theme-based --filter-border-focus Focus border color Theme-based --filter-border-radius Border radius Theme-based --filter-text-color Text color Theme-based --filter-text-active Active text color Theme-based --filter-disabled-opacity Disabled state opacity Theme-based Badge Tokens ■ Token Purpose Default Value --filter-badge-background Badge background color Theme-based --filter-badge-text Badge text color Theme-based --filter-badge-border-radius Badge border radius Theme-based Panel Tokens ■ Token Purpose Default Value --filter-panel-background Panel background color Theme-based --filter-panel-border Panel border style Theme-based --filter-panel-shadow Panel shadow Theme-based --filter-header-background Header background color Theme-based --filter-header-border Header border style Theme-based --filter-header-padding Header padding Theme-based Token Override Example ■ /\* Custom filter theming \*/ .my-custom-filter { --filter-background-primary : #f8fafc ; --filter-border : 1 px solid #e2e8f0 ; --filter-border-radius : 8 px ; --filter-panel-shadow : 0 20 px 25 px -5 px rgba ( 0 , 0 , 0 , 0.1 ) ; } .my-compact-filter { --filter-header-padding : 12 px 16 px ; --filter-badge-border-radius : 12 px ; } Best Practices ■ Design Guidelines ■ Logical Grouping : Organize filters into meaningful, related groups Clear

Labels : Use descriptive, concise labels for filter options Consistent Sizing : Choose appropriate size variants for your interface context Positioning : Consider layout constraints when choosing left/right positioning Badge Usage : Show active filter count for better user awareness Accessibility ■ Keyboard Navigation : Ensure full keyboard support for all interactions Screen Reader Support : Provide clear labels and state announcements Focus Management : Proper focus handling when panel opens/closes Color Contrast : Maintain sufficient contrast for all text and interactive elements ARIA Labels : Use appropriate ARIA attributes for filter groups and options Performance ■ OnPush Strategy : Component uses OnPush change detection for optimal performance Efficient Rendering : Minimal re-renders during filter state changes Memory Management : Proper cleanup of event listeners and references Large Datasets : Consider pagination or virtualization for very large filter groups User Experience ■ Selection Modes : Choose between single and multi-selection based on use case Auto-close : Use auto-close for single-selection to streamline workflow Apply Button : Show apply button when immediate application isn't desired Clear All : Always provide clear all functionality for better usability Visual Feedback : Clear indication of active filters and selection states Integration ■ State Management : Integrate with your application's state management system Event Handling : Use the comprehensive event system for all filter interactions Form Integration : Coordinate with form validation and submission logic Data Binding : Properly bind filter data to your data source Responsive Design : Ensure filter works well on mobile and desktop devices Responsive Behavior ■ Mobile Adaptations ■ The filter component automatically adapts to mobile screens: Touch Optimization : Optimized touch targets for mobile interaction Responsive Sizing : Appropriate sizing for mobile viewports Mobile Positioning : Smart positioning to avoid viewport edges Touch Gestures : Support for touch-based interactions Breakpoint Behavior ■ Desktop (>768px) : Full filter interface with all features Mobile ( $\leq$ 768px) : Compact layout with optimized spacing Panel Sizing : Responsive panel width and height Icon Sizing : Appropriate icon sizes for different screens Content Considerations ■ Group Layout : Filter groups adapt to different screen widths Option Display : Options maintain readability on small screens Button Sizing : Adequate touch target sizes for mobile Text Scaling : Appropriate text sizes for different screen sizes

```

<aava-filter
  size="md"
  title="Filter Products"
  [filterGroups]="filterGroups"
  [showBadge]="true"
  (filterChange)="onFilterChange($event)"
  (clearAll)="onClearAllFilters()"
></aava-filter>

<div class="product-card" *ngFor="let product of filteredProducts">
  <h4>{{ product.name }}</h4>
  <p>Category: {{ product.category }}</p>
  <p class="price">${{ product.price }}</p>
</div>

---

filterGroups = [
  {
    id: 'category',
    title: 'Category',
    options: [
      { id: 'electronics', label: 'Electronics', value: 'electronics' },
      { id: 'clothing', label: 'Clothing', value: 'clothing' },
      { id: 'books', label: 'Books', value: 'books' },
      { id: 'home', label: 'Home & Garden', value: 'home' },
    ],
  },
  {
    id: 'price',
    title: 'Price Range',
    options: [
      { id: 'under-50', label: 'Under $50', value: { min: 0, max: 50 } },
      { id: '50-100', label: '$50 - $100', value: { min: 50, max: 100 } },
      {
        id: '100-200',
        label: '$100 - $200',
        value: { min: 100, max: 200 },
      },
    ],
  },
];
products = [
  { id: 1, name: 'Laptop', category: 'electronics', price: 999 },
  { id: 2, name: 'Smartphone', category: 'electronics', price: 699 },
  { id: 3, name: 'T-Shirt', category: 'clothing', price: 25 },
  { id: 4, name: 'Jeans', category: 'clothing', price: 45 },
  { id: 5, name: 'Novel', category: 'books', price: 15 },
  { id: 6, name: 'Cookbook', category: 'books', price: 35 },
  { id: 7, name: 'Coffee Table', category: 'home', price: 299 },
  { id: 8, name: 'Garden Chair', category: 'home', price: 89 },
];
filteredProducts = [...this.products];

```

```

activeFilters: Record<string, FilterOption[]> = {};

onFilterChange(filters: Record<string, FilterOption[]>): void {
    console.log('Filter changed:', filters);
    this.activeFilters = filters;
    this.applyFilters();
}

onClearAllFilters(): void {
    console.log('All filters cleared');
    this.activeFilters = {};
    this.filteredProducts = [...this.products];
}

private applyFilters(): void {
    this.filteredProducts = this.products.filter((product) => {
        // Category filter
        if (
            this.activeFilters['category'] &&
            this.activeFilters['category'].length > 0
        ) {
            const categoryMatch = this.activeFilters['category'].some(
                (f: FilterOption) => f.value === product.category
            );
            if (!categoryMatch) return false;
        }

        // Price filter
        if (
            this.activeFilters['price'] &&
            this.activeFilters['price'].length > 0
        ) {
            const priceMatch = this.activeFilters['price'].some(
                (f: FilterOption) => {
                    const priceRange = f.value as PriceRange;
                    return (
                        product.price >= priceRange.min && product.price <= priceRange.max
                    );
                }
            );
            if (!priceMatch) return false;
        }

        return true;
    });
}

getActiveFiltersCount(): number {
    let count = 0;
    Object.values(this.activeFilters).forEach((group) => {
        if (Array.isArray(group)) {
            count += group.length;
        }
    });
    return count;
}

```

```
getSelectedCategories(): string {
  const categories = this.activeFilters['category'];
  if (!categories || categories.length === 0) return 'None';
  return categories.map((c: FilterOption) => c.label).join(', ');
}

getSelectedPriceRanges(): string {
  const prices = this.activeFilters['price'];
  if (!prices || prices.length === 0) return 'None';
  return prices.map((p: FilterOption) => p.label).join(', ');
}
```

```

<aava-filter
  size="md"
  title="Filter Users"
  [filterGroups]="filterGroups"
  [showBadge]="true"
  [showApplyButton]="true"
  (filterChange)="onFilterChange($event)"
  (clearAll)="onClearAllFilters()"
  (apply)="onApplyFilters($event)"
></aava-filter>

<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Status</th>
      <th>Role</th>
      <th>Department</th>
      <th>Experience</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let user of filteredUsers">
      <td>{{ user.name }}</td>
      <td>
        <span class="status-badge" [ngClass]="'status-' + user.status">
          {{ user.status }}
        </span>
      </td>
      <td>{{ user.role }}</td>
      <td>{{ user.department }}</td>
      <td>{{ user.experience }}</td>
    </tr>
  </tbody>
</table>

```

---

```

filterGroups = [
  {
    id: 'status',
    title: 'Status',
    options: [
      { id: 'active', label: 'Active', value: 'active' },
      { id: 'inactive', label: 'Inactive', value: 'inactive' },
      { id: 'pending', label: 'Pending', value: 'pending' },
      { id: 'archived', label: 'Archived', value: 'archived' },
    ],
  },
  {
    id: 'role',
    title: 'Role',
    options: [
      { id: 'admin', label: 'Admin', value: 'admin' },

```

```

        {
          id: 'user', label: 'User', value: 'user' },
          {
            id: 'moderator', label: 'Moderator', value: 'moderator' },
            {
              id: 'guest', label: 'Guest', value: 'guest' },
            ],
        },
      },
      {
        id: 'department',
        title: 'Department',
        options: [
          { id: 'it', label: 'IT', value: 'IT' },
          { id: 'hr', label: 'HR', value: 'HR' },
          { id: 'finance', label: 'Finance', value: 'Finance' },
          { id: 'marketing', label: 'Marketing', value: 'Marketing' },
          { id: 'sales', label: 'Sales', value: 'Sales' },
        ],
      },
    },
    {
      id: 'experience',
      title: 'Experience Level',
      options: [
        { id: 'junior', label: 'Junior (0-2 years)', value: 'junior' },
        { id: 'mid', label: 'Mid-level (3-5 years)', value: 'mid' },
        { id: 'senior', label: 'Senior (6-10 years)', value: 'senior' },
      ],
    },
  ],
};

users = [
  {
    id: 1,
    name: 'John Doe',
    status: 'active',
    role: 'admin',
    department: 'IT',
    experience: 'senior',
  },
  {
    id: 2,
    name: 'Jane Smith',
    status: 'active',
    role: 'user',
    department: 'HR',
    experience: 'mid',
  },
  {
    id: 3,
    name: 'Bob Johnson',
    status: 'inactive',
    role: 'moderator',
    department: 'Finance',
    experience: 'senior',
  },
  {
    id: 4,
    name: 'Alice Brown',
    status: 'active',
  }
];

```

```
        role: 'user',
        department: 'Marketing',
        experience: 'junior',
    },
{
    id: 5,
    name: 'Charlie Wilson',
    status: 'pending',
    role: 'user',
    department: 'IT',
    experience: 'mid',
},
{
    id: 6,
    name: 'Diana Davis',
    status: 'active',
    role: 'admin',
    department: 'HR',
    experience: 'expert',
},
{
    id: 7,
    name: 'Evan Miller',
    status: 'archived',
    role: 'guest',
    department: 'Sales',
    experience: 'junior',
},
{
    id: 8,
    name: 'Fiona Garcia',
    status: 'active',
    role: 'moderator',
    department: 'IT',
    experience: 'senior',
},
];
filteredUsers = [...this.users];
activeFilters: Record<string, FilterOption[]> = {};

onFilterChange(filters: Record<string, FilterOption[]>): void {
    console.log('Filter changed:', filters);
    this.activeFilters = filters;
    this.applyFilters();
}

onClearAllFilters(): void {
    console.log('All filters cleared');
    this.activeFilters = {};
    this.filteredUsers = [...this.users];
}

onApplyFilters(filters: Record<string, FilterOption[]>): void {
    console.log('Filters applied:', filters);
    this.activeFilters = filters;
}
```

```

        this.applyFilters();
    }

private applyFilters(): void {
    this.filteredUsers = this.users.filter((user) => {
        // Status filter
        if (
            this.activeFilters['status'] &&
            this.activeFilters['status'].length > 0
        ) {
            const statusMatch = this.activeFilters['status'].some(
                (f: FilterOption) => f.value === user.status
            );
            if (!statusMatch) return false;
        }

        // Role filter
        if (this.activeFilters['role'] && this.activeFilters['role'].length > 0) {
            const roleMatch = this.activeFilters['role'].some(
                (f: FilterOption) => f.value === user.role
            );
            if (!roleMatch) return false;
        }

        // Department filter
        if (
            this.activeFilters['department'] &&
            this.activeFilters['department'].length > 0
        ) {
            const deptMatch = this.activeFilters['department'].some(
                (f: FilterOption) => f.value === user.department
            );
            if (!deptMatch) return false;
        }

        // Experience filter
        if (
            this.activeFilters['experience'] &&
            this.activeFilters['experience'].length > 0
        ) {
            const expMatch = this.activeFilters['experience'].some(
                (f: FilterOption) => f.value === user.experience
            );
            if (!expMatch) return false;
        }

        return true;
    });
}

getActiveFiltersCount(): number {
    let count = 0;
    Object.values(this.activeFilters).forEach((group) => {
        if (Array.isArray(group)) {
            count += group.length;
        }
    })
}

```

```
    });
    return count;
}

getFilterSummary(): string {
  const summaries: string[] = [];

  if (this.activeFilters['status']?.length) {
    summaries.push(` ${this.activeFilters['status'].length} status(es)`);
  }
  if (this.activeFilters['role']?.length) {
    summaries.push(` ${this.activeFilters['role'].length} role(s)`);
  }
  if (this.activeFilters['department']?.length) {
    summaries.push(
      ` ${this.activeFilters['department'].length} department(s)`
    );
  }
  if (this.activeFilters['experience']?.length) {
    summaries.push(
      ` ${this.activeFilters['experience'].length} experience level(s)`
    );
  }

  return summaries.length > 0 ? summaries.join(', ') : 'No filters applied';
}
```

```

<aava-filter
  [singleSelection]="true"
  title="Sort By"
  [filterGroups]="sortFilterGroups"
  [showClearAll)="true"
  [showApplyButton]="false"
  size="md"
  (singleSelectionChange)="onSortChange($event)"
></aava-filter>

<div class="product-card" *ngFor="let product of sortedProducts; let i = index">
  <span class="order-number">#{{ i + 1 }}</span>
  <h4>{{ product.name }}</h4>
  <p>Category: {{ product.category }}</p>
  <p class="price">${{ product.price }}</p>
  <p class="date">{{ product.date | date : "shortDate" }}</p>
</div>

---

sortFilterGroups = [
  {
    id: "sortBy",
    title: "Sort By",
    options: [
      { id: "name-asc", label: "Name (A-Z)", value: "name_asc" },
      { id: "name-desc", label: "Name (Z-A)", value: "name_desc" },
      { id: "price-low", label: "Price (Low to High)", value: "price_asc" },
      { id: "price-high", label: "Price (High to Low)", value: "price_desc" },
      { id: "newest", label: "Newest First", value: "date_desc" },
      { id: "oldest", label: "Oldest First", value: "date_asc" },
    ],
  },
];
products = [
  {
    id: 1,
    name: 'Laptop',
    category: 'electronics',
    price: 999,
    date: '2024-01-15',
  },
  {
    id: 2,
    name: 'Smartphone',
    category: 'electronics',
    price: 699,
    date: '2024-01-10',
  },
  {
    id: 3,
    name: 'T-Shirt',
    category: 'clothing',
    price: 25,
  }
];

```

```
        date: '2024-01-20',
    },
{
    id: 4,
    name: 'Jeans',
    category: 'clothing',
    price: 45,
    date: '2024-01-18',
},
{ id: 5, name: 'Novel', category: 'books', price: 15, date: '2024-01-12' },
{
    id: 6,
    name: 'Cookbook',
    category: 'books',
    price: 35,
    date: '2024-01-14',
},
{
    id: 7,
    name: 'Coffee Table',
    category: 'home',
    price: 299,
    date: '2024-01-16',
},
{
    id: 8,
    name: 'Garden Chair',
    category: 'home',
    price: 89,
    date: '2024-01-19',
},
{
    id: 9,
    name: 'Running Shoes',
    category: 'sports',
    price: 120,
    date: '2024-01-17',
},
{
    id: 10,
    name: 'Yoga Mat',
    category: 'sports',
    price: 45,
    date: '2024-01-13',
},
];
sortedProducts = [...this.products];
selectedSort: FilterOption | null = null;
selectedCategory: FilterOption | null = null;
selectedStatus: FilterOption | null = null;

onSortChange(selection: Record<string, FilterOption | null>): void {
    const sortOption = selection['sortBy'];
    this.selectedSort = sortOption;
    console.log('Sort changed:', sortOption);
```

```

    if (sortOption) {
      this.applySorting(sortOption.value);
    }
  }

private applySorting(sortBy: string): void {
  this.sortedProducts = [...this.products];

  switch (sortBy) {
    case 'name_asc':
      this.sortedProducts.sort((a, b) => a.name.localeCompare(b.name));
      break;
    case 'name_desc':
      this.sortedProducts.sort((a, b) => b.name.localeCompare(a.name));
      break;
    case 'price_asc':
      this.sortedProducts.sort((a, b) => a.price - b.price);
      break;
    case 'price_desc':
      this.sortedProducts.sort((a, b) => b.price - a.price);
      break;
    case 'date_desc':
      this.sortedProducts.sort(
        (a, b) => new Date(b.date).getTime() - new Date(a.date).getTime()
      );
      break;
    case 'date_asc':
      this.sortedProducts.sort(
        (a, b) => new Date(a.date).getTime() - new Date(b.date).getTime()
      );
      break;
  }
}

getSelectedSortLabel(): string {
  return this.selectedSort ? this.selectedSort.label : 'None';
}

getSelectedCategoryLabel(): string {
  return this.selectedCategory ? this.selectedCategory.label : 'None';
}

getSelectedStatusLabel(): string {
  return this.selectedStatus ? this.selectedStatus.label : 'None';
}

```

■ No code found